REMARKS

This amendment is presented under the provisions of Rule 116(b)in response to the Final Office Action mailed on April 21, 2005, for the purpose of placing the application for allowance or in the alternative, in better form for Appeal. Claims 1-22 are active in the application.

Applicants have made minor changes to the claims. The amendments to the claims were made for the purpose of consistency in the steps set forth in the claims. Such amendments do not raise new issues or necessitate a new search and therefore, should be entered.

Applicants have reviewed the final rejection and find that it essentially puts forth the same grounds of rejection as the first Office Action. Applicants have previously addressed these grounds of rejection in their previous amendment. Therefore, these arguments will only be repeated as necessary herein.

THE FINAL REJECTION SHOULD BE DEEMED PREMATURE

Relative to this point, Applicants find it most important to address the Examiner's response to Applicants arguments. The Examiner's response reveals an important issue regarding the proper interpretation of the Lam patent and what seems to be an overly broad interpretation of Applicants claims.

MISSING APPENDICES

In interpreting the Lam patent, Applicants submit that it has become very important to know what information is contained in the Appendices cited and incorporated by reference into the patent. As suggested in Applicants previous amendment, the absence of the Appendices was considered and noted. Applicants were of the opinion at that time that the distinctions between RPC procedure messaging and Applicants record blocking set forth in the amendment would be deemed sufficient. Unfortunately, this was not the case. Applicants have made efforts to obtain the omitted Appendix material. It was advised that the Appendices could have been provided as part of the patent at the option of the Examiner, but this was not done. The USPTO legal office was contacted and a portion of Appendix A was provided. The portion corresponding to pages 30-33 was reproduced in part and is enclosed herewith.

It can be seen that Appendix A lists a number of IPC requests and apparently, the structure of each request is set forth in the remaining part of the Appendix. Applicants still are unable to obtain a copy of the complete appendix in a timely fashion. It is submitted that the Examiner is in the best position to obtain a copy of the Appendix and make it available to Applicants. In view of this, Applicants submit that the final rejection should be deemed premature because Applicants have not been afforded a fair and complete opportunity to present arguments as to the **complete teachings** of Lam satisfactory to the Examiner in view of the missing appendices and how Applicants claims distinguish patentably over such teachings. The Examiner will note that the Appendix could not be obtained to be presented earlier and up until now the significance of its importance established.

CLAIMS ARE BEING INTERPRETED TOO BROADLY

Applicants further believe that the claims are being interpreted too broadly. As discussed in section 2111 of the Manual of Patent Examination, the claims are to be read in light of the specification to interpret explicit limitations recited in the claims. This differs from reading limitations disclosed in the specification that have no express basis in the claims. There are specific express recitations in the claims which are to be interpreted in light of the specification and that should not be ignored.

The Examiner indicates in his response to Applicants arguments that the features upon which applicant relies (i.e. messages not being equivalent to records) are not recited in the rejected claims. The Examiner has not set forth the basis for this comment. For example, claim 1 contains several steps that specifically limit the claim to an arrangement requiring records that are blocked and unblocked. Further, record blocking occurs via specific APIs. Applicants submit that the meaning of these words used in the claims can not be ignored. Applicant's specification discusses that meaning of records, blocked and unblocked in various sections. Applicants have included an attachment that contains relevant portions of the specification that utilize these terms. The attachment also indicates the section that makes reference to the term messages.

Applicants submit that the Examiner has not set forth an explanation indicating the basis for the Examiner's position. Because of this lack of explanation which would enable Applicants to respond to the final rejection, this should be another reason as to why the final rejection should be deemed premature particularly since Applicants have provided specific evidence as to the significance of the terms used in the claims

It should be noted that the teachings of Lam recognize distinctions between similar terms. For example, Lam recognizes the differences between commands that are blocked from procedural calls. Lam cites a DMI specification which is distinguished relative to Figure 3 of Lam in terms of disclosing a management interface (MI) 310 used by management applications (programs that initiate management requests) as being a data interface (**command** blocks) as opposed to the procedural interface used in a RPC.

According to Lam, data blocks describe the format for data transfer instead of parameters to a function call. The component interface is also a data interface. It is used by component providers to describe access to management information and to **enable a component to be managed**. In the early DMI specification, the CI was defined to be a block oriented data interface as opposed to a procedural interface. This was changed in the present DMI specification. The management interface was also changed to a procedural interface. It is a remoteable interface designed to be used with one of the supported RPCs. In the early DMI specification, both MI and CI were local interfaces. Thus, it is seen that Lam distinguishes a block oriented transfer of commands from a procedural interface that transfers parameters to a function call. This indicates that these terms have different distinct meanings to those skilled in the art. To attempt to interpret such terms broadly to the point of saying that they are equivalent is contrary how a skilled artisan would interpret such terms. The same applies to attempting to equate terms such as "blocked records" and "messages".

THE LAM PATENT IS DIRECTED TO AN ENTIRELY DIFFERENT PROBLEM

The Examiner responded to Applicants arguments regarding claim 1 that the Lam patent does not disclose a method of translating blocked data transferred from a program by citing column 8, lines 18-32 and column 9, lines 46-55 as more explicit explanations of message format conversion. Before discussing the cited material, it is helpful to discuss claim 1.

Claim 1

Claim 1 is directed to a method of translating blocked data transferred from a first program executing on a first computer system to a second computer system wherein the computer systems are heterogeneous computer systems coupled together over a communications link. The method employs an API by which the first program communicates with a first interface. In the preferred embodiment, the first program (written in COBOL) calls specific procedure functions of the API (Connection Manager component 170) that provides a **record oriented connection** between the programs using it (see description in item 6 of the Attachment).

In the preferred embodiment, the first function called by the first program is an **Open** function that establishes a connection to the second computer using the sockets interface. This operation corresponds to step (A) of claim 1. Next, as indicated in step (B), the first program of the preferred embodiment specifies a first translation for records transmitted over the first session. In the preferred embodiment, the first program calls a function **Define Records** that assigns a record definition that specifies the data conversions that subsequent read and write procedures perform (see description in section 5.1.11 of item 6 of the Attachment). As discussed at page 10 of the specification, in the preferred embodiment, since the data transfers between systems is typically on a (blocked) record basis, this data conversion can be selected on a per field basis and is performed on each selected field in each record transferred (see second paragraph of item 1 of the Attachment).

As indicated in step (C) of claim 1, the first computer blocks a first plurality of records into a first block of records. In the preferred embodiment, the first program calls a function **Write a Record** within a Record Manager component which moves a record into a collection buffer for the connection (see section 3.1.2 of item 3 and section 5.1.7 of item 6 of the Attachment). The first program repeatedly calls this Record Manager function resulting in the filling of the collection buffer.

As indicated in step (D), the first block of records is transmitted over the first session from the first computer to the second computer. In the preferred embodiment, the record manager component communicates with a Sockets Interface (see item 1 of the Appendix). It will be noted that it is **this socket interface** that performs what could be termed messaging. As discussed herein, in Lam, it is the RPC interface that performs messaging. In the case of the present invention, since the blocking of records is done **prior to the call** to the sockets interface, there is no need to perform repeated communications with the sockets interface for each record (see section 4.1.3 of item 5 of the Attachment). This results in a substantial performance improvement. When the sockets interface is used in the preferred embodiment, the buffer is encapsulated by a socket header. Thus, the only part of claim 1 that could be said to relate to messaging is step (D) which takes place after the blocking of records.

In step (E), the second computer unblocks the first block of records. In the preferred embodiment, this is carried out by a Record Manager component 182. In step (F), each record is translated in accordance with the translation specified in step (B). As stated in the Attachment, in the preferred embodiment, the conversion is performed on the first system by the Record Manager component. But such conversions can be performed by the Record Manager component on the second computer. In both cases, the Record Manager component uses the record definition obtained by the first program as discussed above. Thus, it is seen that claim 1 is directed to providing a record connection API by which a first program (i.e. in the preferred embodiment a COBOL program) issues functions which result in the blocking of data records and establish how the different fields of each record are to be translated.

This is to be contrasted with the teachings of the Lam patent discussed herein.

The Lam Patent Invention

The problem that Lam sets out to **solve** is to eliminate the problems in supporting **different versions of APIs** used by **remote procedure call** (RPC) modules on client and server computers. Lam seeks to eliminate the prior art problems associated with **incompatible versions** of **function calls** supported by remote client and server **RPC** command modules 515 and 525 over network 500. Lam provides that the RPC modules support only a single RPC command and not every single command in the list of procedure calls for every possible version. This single command transfers a buffer of information which does not affect the operation of the RPC module.

More specifically, Lam uses API component 512 to provide a specific API function call. The call is a single message transfer remote procedure call to the RPC command module that causes the transfer of the buffer of information that does not affect the operation of the RPC module. The buffer is filled with message request information (specified in Appendix A by Lam not provided by the Examiner as discussed above) which is transferred to the second computer. As discussed above, the portion of the Appendix provided by the USPTO, the message is one of a number of IPCs having a specific format or structure.

As discussed in Lam, the API component 512 does not directly provide component management function calls with parameters to remote client RPC command module 515 as in the prior art RPC. Instead, the API component 512 generates a message RPC_MESSAGE_REQUEST that identifies the called function and version of the remote client API 512. The API 512 calls a local message transfer RPC command to send the message to the RPC command module 515.

The RPC command module 515 supports a first set of operations that processes the local message transfer RPC command and **packages** the message for transfer as a **RPC** over network 500. The message is packaged using the prior art techniques for the particular type of RPC implementation on the remote client computer 510. The packaged message is then sent over the network by the network stack 516 to the server computer.

The server network stack 526 provides the packaged RPC to server RPC command module 525. In the case of the local message transfer RPC command, message RPC_MESSAGE_REQUEST is passed to the server API 522 which **parses** the message to determine the **computer architecture** of the remote client computer 510 (e.g. addressing format). With this information, the server API 522 reads the version specified in the message and compares the two. If the client API version is incompatible with the server API 522, the server sends a reply indicating the version incompatibility. If the two are compatible, then the message RPC_MESSAGE_REQUEST is passed to I/O manager 530 as message IPC_MESSAGE in a format compatible with the I/O manager 530, using standard operating system interprocess communication (IPC). It is the responsibility of **the server** API 522 to assure that the message RPC_MESSAGE_REQUEST **is converted** to message IPC_MESSAGE in a form (e.g. addressing format) that can be processed by I/O manager 522.

As discussed in Lam, the message contained in the buffer that is transferred to the server must be in a specific byte order format in order that the computer architecture field can be read correctly independently of the addressing method used by a particular computer architecture (it is **one byte aligned**). When the request is in a different format than that processed by the I/O manager server 631 of API 522, module 623 converts the format of the message to a **neutral canonical format IPC message**. Thereafter, the server module 623 performs the version compatibility testing required for proper operation. According to Lam, the **conversion** of the message at the receiving end is advantageous in that **when necessary**, the conversion of commands to I/O manager 530 **is done only on the server computer** which is typically more powerful than the client computer (see column 12, lines 26-65).

In response to the IPC message (e.g. command-function call), the I/O manager server for interprocess communication 631 issues a call to the appropriate management function in the I/O manager server 632. The server 632 performs the called management function for the specified component and returns the result to the requesting client.

From this description, it is seen how the subject matter of claim 1 is distinguishable from the teachings of Lam in terms of the problem being solved and the described solution. It is seen from the above that Lam requires his solution fit into an RPC environment requiring (1) Component Management, (2) RPC Client and Server Command Modules, I/O Manager and Specific Components to handle various function calls. By contrast, the solution of the present invention just involves functions provided via an API from a program which in the preferred embodiment is a normally written COBOL program.

<u>The cited material of columns 8 and 9</u>

In rebuttal of Applicants argument that Lam does not disclose a method of translating blocked data transferred from a program, the Examiner cites material that discusses the conversion of format of the message IPC_MESSAGE discussed above and conversion of an event message EV_MESSAGE. In both cases, the conversion is **conditional** in that it only takes place if it is necessary. It will be noted that an event message is **generated** by the server and transmitted only to registered clients. It is unsolicited and has nothing to do with a request message generated by a client computer.

Applicants still submit that the cited material does not disclose a method of translating blocked data transferred from a program as set forth in claim 1. The above described message conversion is performed on the format of the message and converts the message to a neutral canonical format IPC message. Again, it should be noted that the IPC message contains a command which is to be executed by the server. It is not data to be written as in the case of claim 1. Moreover, it is certainly not in record form as required by claim 1.

In view of the above discussion, it can be seen that the previously rejected following claims should be deemed patentable over the cited teachings of Lam. These distinctions are repeated for the convenience of the Examiner.

Claim 2

For the reasons given above relative to claim 1, Lam is absent a teaching of performing a translation step by a first interface as specified in claim 2. Lam is absent a translation step relative to a first plurality of records as discussed above. The conversion of a message format is conditional and pertains to **commands** (IPC requests). Further, Lam teaches that such conversion is done only on the server computer. For these reasons, claim 2 should be deemed patentable over the teachings of Lam.

Claim 3

For the reasons given above relative to claim 1, Lam is absent a teaching of performing a translation step of **records** by a second interface. The material cited in column 6, lines 12-16 cited by the Examiner pertains to conversion of an **addressing format** of a **message** compatible with the server component. By contrast, claim 3 is directed to translation of records (i.e. the data contained in the fields of such records) specified by a first program.

Claim 5

As discussed above, Lam is not concerned with records let alone records, each of which comprise a plurality of fields and the translation of an integer format from a first format to a second format. The material previously cited by the Examiner in column 12, lines 44-49 concerns the **conditional** translation of a **message request** that is in an address format that is different from that processed by the I/O manager server 631. As previously discussed above, Lam provides that the server component convert the format to a neutral canonical format message. Lam provides an example of this which corresponds to the material cited by Examiner-that of converting a message in a little endian **addressing** format to a big endian **addressing** format. Again, Lam is concerned with addressing formats of messages and not data record representations. Also, Lam is not concerned with translating an integer in **one** of the plurality fields in a record from a first integer format to a second integer format. Clearly, Lam does not provide any mechanism that enables data conversion of a selected field of data record which is performed on each record transferred.

<u>Claim 6</u>

For reasons given above, Applicants find Lam absent translating an integer from a first endian format to a second endian format as specified in claim 6. Again, Lam is concerned with addressing formats of messages and not data formats of records as defined in claim 6. Further, as stated relative to claim 5, Lam does not provide data conversion of a selected field of each data record.

<u>Claim 8</u>

Applicants find Lam absent step B, let alone disclosing the use of a file containing a record description as recited in claim 8. The material cited by the Examiner contained in column 5, lines 35-42 pertains to a field read in a message that includes an identifier of the computer architecture of the first computer as discussed above. The identifier is used to determine if the addressing formats/architectures of the client and server computers are the same. Applicants submit that the use of an identifier specifying the addressing format of a computer should not be deemed the equivalent of using a file containing a record description for specifying the particular type of translation via an API for designated fields within a record according to claim 8.

<u>Claim 9</u>

For reasons given with respect to claim 8, Applicants find Lam absent the specifying step (B) via an API let alone utilizing a memory area containing a record description as specified in claim 9. The material cited by the Examiner at column 5, lines 47-53 pertains to the building/packing of the message in a buffer memory. The message includes an identifier for the called component management function and a version of the component management application programming interface. The version identifier is used to determine if the client and server APIs are compatible as discussed above. Thus, this arrangement is distinctively different to that recited in claim 9.

__Claims 10-13, 15, 16 and 18-22__

For the reasons given above, claims 10-13, 15, 16 and 18-22 should also be deemed patentable over the cited teachings of Lam. Further, claims 10 and 20 specify the opening of a second session from the first program. For the reasons discussed above relative to claim 1, Applicants submit that Lam does not teach the opening of a first session let alone the steps specified in claims 10 and 20.

__Claim Rejection - 35 U.S.C. § 103__

The Examiner responded to Applicants argument that there is no suggestion to combine the references by stating that the motivation to do so can be found in the knowledge generally available to one of ordinary skill in the art. But, the Examiner does not set forth with clarity the particular knowledge that would so motivate a skilled artisan. The Examiner cites In re Fine in support of his position. In the cited decision of Fine, the Court stated that "To imbue one of ordinary skill in the art with knowledge of the invention in suit, when no prior art reference or references of record convey or suggest that knowledge, is to fall victim to the insidious effect of a hindsight syndrome wherein that which only the inventor taught is used against the teacher." Generally, the rationale for combining references is a recognition, expressly or impliedly in the prior art or drawn from a convincing line of reasoning based on established scientific principles or legal precedent that some advantage or expected beneficial result would have been produced by their combination. (see MPEP section 2144)

In the prior Office Action, the Examiner indicated that it would have been obvious to one of ordinary skill in the art to modify Lam in view of Allen to translate a first character format to a second character format and translate a first floating point format to a second floating point format. The Examiner stated that one would be motivated to do so because it would allow **for translation of different types of data**. However, the Examiner has presented no particular line of reasoning as to why this capability would benefit the teachings of Lam. Applicants find no discussion in Lam of any need to provide for the translation of different types of data. There is only need to translate the addressing message format of the message request when they are not the same. This translation is so that the server computer can process the message request when it has been established that the APIs of the two computers are the same.

In view of this, Applicants find a lack of motivation since the Examiner has not presented an actual line of reasoning for his conclusion. Further, Applicants find no reason as to why would a system such as Lam that involves RPC requests relative to the management of components would be concerned about providing for translation of different types of data. As discussed herein, the teachings of Allen also suggest non-combination with those of Lam.

<u>TEACHINGS OF ALLEN PATENT</u>

Allen overcomes drawbacks to the multitude of computer systems designed differently, the way that data is actually stored on each computer system may be different. If the data changes in some way, the program must be modified or rewritten to deal with the changes. Allen is directed to solving the problems of having to rewrite and modify data conversion programs.

Allen provides a generic data converter used by a client application 123 that uses a data description (text file) **describing the input and output data** sent to and received from a server program 195. With the invention according to Allen, instead of hard-coding a program to find and convert all of the data elements, a software engineer can create a data description that describes the data elements and their interrelationships. The client application calls the data converter with the identification (name) of the server program. The invention is used in a client-server architecture in which vast amounts of highly complex and structured data are being transferred between dissimilar computers. The data could be a file, a data stream, an object, a structure or pointer thereto. Instead of rewriting large amounts of "hard coded" software, changes to the data description 124 may be made that describe the changes to the output data, the interrelationship between the data, or how the data is to be converted. In addition, changes to input data and how the data is to be converted or the input parameter list used to call a server program may also be easily and conveniently updated. Such conversion is performed **only by the client** application. The conversion is performed "**just-in-time**" which should be faster because unused or undesired data will not be converted. This is in contrast to "hard-coded" conversion programs, which convert all data whether requested or not. That is, the data converter waits until the client application asks for particular data elements in the received data before converting the data.

3. DIFFERENCES IN PURPOSE AND MODE OF OPERATION SUGGEST NON-COMBINATION OF THE TEACHINGS OF LAM AND ALLEN

In Lam, the problem being solved is to eliminate the need for rewriting RPC command modules when there are changes in APIs and to simplify such modules. Allen solves the problem of having to rewrite and modify data conversion programs when the data changes in some way. In Allen, the conversion of data is done at the client end when the data is requested by the client application.

In Lam, the conversion of commands is done at the receiving server end. That is, as discussed above, it is the responsibility of the server API 522 to assure that the message RPC_MESSAGE_REQUEST **is converted** to message IPC_MESSAGE in a form (e.g. addressing format) that can be processed by I/O manager 522. That is, the conversion must take place in order for the server to process the request/command. If conversion is to be done "just in time" and by the client system as taught by Allen, this would render the Lam system inoperative. Applicants submit that it is not obvious to modify the prior art Lam system which would lead to render such system inoperable for its intended purpose (see In re Gordon 221 USPQ 1125 Fed. Cir. 1984). Additionally, for the above reasons, Applicants submit that the references teach away from their combination.

As discussed above, Applicants found no motivation to combine Lam and Allen other than that of Applicants invention. It could be said that the basis would be that it was obvious to try to include the teachings of Allen in the Lam system which is an improper basis for rejection (see In re Fine cited by the Examiner). Applicants submit that there is no knowledge available that would prompt a skilled artisan to combine the teachings of a message based system which sends RPC commands for the purpose of managing components with the teachings of a system that employs data conversion in object oriented programming, Java and XML technologies.

First of all, the material cited in Lam by the Examiner does not pertain to the data formats or types but rather addressing formats of computers relative to transfer of messages between computers. Lam provides conversion of messages into a neutral canonical format message. Once such conversion of messages takes place, the messages can be processed by the receiving server system. Other than the Examiner's statement, Applicants submit that there is no reason to provide additional conversions or different conversions to accomplish the objectives of the Lam patent. The material cited by the Examiner in Allen pertains to document processing and expediting the parsing of the data elements contained in such documents by a generic data converter through the use of a data description (XML parser output that has been placed into a hash table object).

Neither Lam nor Allen teach the use of data records, let alone records that comprise a plurality of fields, one of which is an alphanumeric field and the translation of each character in one such selected field from a first character format to a second character format as defined in claim 4. Similarly, relative to claim 7, neither Lam nor Allen teach the use of data records, let alone records that comprise a plurality of fields, one of which is a floating point field and the translation of each floating point numbers in that one **selected** field from a first floating point format to a second floating point format.

The material cited by the Examiner in column 16 discusses the converting of character sets through the use of additional description to a description of hostName and the use of a code page that allows the data converter to know what language the characters are in the data and what and how to convert them to the equivalent language in Unicode. Clearly, since Lam and Allen are directed to entirely different types of schemes, Applicants submit that there would be no motivation to try to combine such teachings, let alone modify Lam in the manner specified by the Examiner.

For similar reasons, claims 14 and 17 should also be deemed patentable over the proposed combination of Lam and Allen. Further, claim 17 specifies carrying out such translation through the use of a set of computer instructions while Allen teaches the use of a generic data converter.

Applicants have shown that the final rejection should be deemed premature for the above reasons pertaining to the non-inclusion of the very pertinent Appendices of the Lam patent, the omission of any basis for concluding that the rejected claims do not recite features discussed in Applicants prior amendment, the overly broad interpretation of the claims-note the recitation of means in claim 22 and the omission of the basis for motivation to combine the teachings of the Lam and Allen patents. Also, Applicants have set forth additional arguments as to why claims 1-22 should be deemed patentable over the cited prior art and should be deemed in condition for allowance. A notice to this effect is respectfully solicited.

Applicants ask the Examiner to contact Applicants attorney upon receipt of this amendment for the purpose of advancing the prosecution of this application.

Also, if any questions or issues should arise with respect to this amendment or this application, the Examiner is **urged** to **call Applicants' attorney at the number indicated herein**. .

Respectfully submitted,

Faith F. Driscoll

Faith F. Driscoll
Registration No. 24,206
Attorney for Applicants
(781) 326-6645

Encl: Attachments
FFD/fd

REPRODUCED PORTION OF LAM PATENT APPENDIX A

An Example of A Message Structure and Related Definitions

```
                              // MESSAGE TYPES
#define IPCREQ              0x00000000          // A Request Message

#define IPCRES              0x00010000          // Result of request

                                //REQUESTS

// DATABASE related

#define IPC_GETNEXTLOGICALCOMP_REQ              (IPCREQl1)
#define IPC_GETNEXTPHYSICALCOMP_REQ             (IPCREQl2)


#define IPC_GETP2LCONN_REQ                      (IPCREQl17

// MDLMGR RELATED

#define IPC_SUBMITJOB_REQ                       (IPCREQl18)
#define IPC_MODIFYSCHED_REQ                     (IPCREQl19)
```

```
#define IPC_CHANGE_UNITNUM_REQ                    (IPCREQ140)

#ifdef ASPIMODEL
#define IPC_ASPIPOLL_REQ                          (IPCREQ 1 64)
#define IPC_ASPIDISCOVER_REQ                      (IPCREQ 1 65)
#define IPC_ASPICLEARDB_REQ                       (IPCREQ 1 66)
#endif

                                //RESULTS
//DATABASE related
#define IPC_GETNEXTLOGICALCOMP_RES                (IPCRES11)
#define IPC_GETNEXTPHYSICALCOMP_RES               (IPCRES12)
```

```
#define IPC_GETP2LCONN_RES                    (IPCRES117)

// MDLMGR RELATED
#define IPC_SUBMITJOB_RES                     (IPCRES 118)
#define IPC_MODIFYSCHED_RES                   (IPCRES 119)


#define IPC_CHANGE_UNITNUM_RES                (IPCRES 140)

#ifdef ASPIMODEL
#define IPC_ASPIPOLL_RES                      (IPCREQ 1 64)
#define IPC_ASPIDISCOVER_RES                  (IPCREQ 1 65)
#define IPC_ASPICLEARDB_RES                   (IPCREQ 1 66)
#endif

        // IPC MESSAGE STRUCTURES

typedef struct {
```

APPENDIX OF SPECIFICATION REFERENCES

1. (Page 9 of Specification) FIG. 9 is a block diagram that illustrates in further detail the modules utilized in a preferred embodiment of the present invention. For the most part, the modules utilized in both heterogeneous computer systems are equivalent, and will be discussed together here. They are discussed more thoroughly below. An application **150** on the first system **110** communicates with a GCOS File Interface Procedure (GFIP) **171**. An application **160** and/or the Unix File Read/Write Application (UFAP) **166** on the second system **112** communicates with a UNIX File Interface Procedure (UFIP) **181**. Both the GFIP **171** and the UFIP **181** are comprised of an ETL Connection Manager (ECM) **170**, **180** which communicate with a Record Manager **172**, **182**, which in turn either communicate with a SID Interface (SIDI) **174**, **184**, in the case of a "Fast Link" **158**, or a Sockets Interface (SOCKI) **176**, **186**, in the case of a communications link **152**.

(Page 10 of Specification) Another improvement has been made to the prior art. The application **130** on the first computer system **110** can specify what data conversions are to be performed by the interface between systems. Since the data transfers between systems is typically on a (blocked) record basis, this data conversion can be selected on a per field basis, and is performed on each selected field in each record transferred. Thus, some fields can be converted automatically from 36-bit integers to 32 bit integers (and potentially reversing the "endian" for the integers at the same time), while other fields can be converted from 9-bit ASCII to 8-bit ASCII.

2. (Page 10 of Specification) In the preferred embodiment, a " Data Transform Request (DTR) File" (see Appendix A for format of this file) is a parameter to an X_ETL_DEFINEREC **API function call** and specifies the conversions that are to be performed. In alternate embodiments, this information is specified by other means, such as by Cobol record formats or a database schema or subschema. Also, in other embodiments, this information can be provided in memory instead of as a file. In the preferred embodiment, this conversion is performed on the first system. However, in other embodiments, this conversion can be performed on the second (UNIX) system.

The preferred embodiment of the present invention consists of a GCOS 8 mainframe computer system as the first computer system 110, and an AIX UNIX computer system as the second computer system **112**. It should be understood that this is illustrative only, and that the present invention includes other heterogeneous computer systems.

3. 3.1.1 ETL Connection Manager (ECM) 170

GFIP's ECM **170** component provides the GCOS 8 user application with the API defined in the Section entitled "EIS COBOL 8 COBOL-85 API.".. ECM **170** manages the Fast-ETL connections established by an application, creates and responds to records with the record-type codes defined hereinbelow, and uses the services of Record Manager **172** to **block and unblock records.**

3.1.1.1 Open A transfer between GCOS 110 and UNIX 112 is typically initiated by the GCOS application calling X_ETL_OPEN. One of the parameters to this function indicates which direction records are being transferred. The following Table 3.1.1.1 illustrates the interaction between GFIP 171 and UFIP 181 for this function:

3.1.2 Record Manager 172

GFIP's Record Manager **172** component is called by ECM **170** and provides ECM **170** with services that are independent of Fast-ETL record-type. These services include:

- packing **records** into buffers and sending these buffers to UFIP
- receiving buffers from UFIP, unpacking the **records** from the buffers, & giving the records to ECM **170**

4. *3.2 UNIX File Interface Procedures (UFIP)*

UFIP **181** is identical in its design to GFIP **171**, which is described above. The code is not identical because of the following differences:

- GFIP employs a VMPS based status return structure. UFIP uses the UNIX convention of an integer status return value.
- UFIP supports a caller specified timeout value. GFIP blocks until UFIP returns an error, or disconnects.

GFIP performs the 9 to 8-bit conversions required for integers. This occurs in the integer fields of the ETL buffer header, in the ETL record header, and in user data for ETLBITS data format. In addition, GFIP calls the data conversion routines required for NCR RDBC support.

5. 4.1.2 GFIP/UFIP Buffer Format

GFIP 171 and UFIP 181 **block records into buffers for efficiency**. In the preferred embodiment, in order to maintain compatibility with the LCB interface, the buffer size is limited to 32k bytes.

In the preferred embodiment, buffers are defined and exchanged in a 9-bit byte format. This allows GFIP 171 to be more efficient in GCOS CPU cycles, with the tradeoff being more UNIX CPU cycles.

The buffer format is defined below in Table 4.1.2.1:

Table 4.1.2.1

Table 4.1.2.1
GFIP/UFIP Buffer Format

Bytes	Width	Field
00-03	04	ID = "ETL " (4 ASCII Characters)
04-07	04	Buffer_length (in 9-bit bytes)
08-11	04	Number of records
12-19	08	Buffer_sequence number
20-27	08	RFU
28-31	04	Debug_options
32-127	96	RFU
128-9999		Records

4.1.3 Fast-ETL Record Types

The data exchanged between GFIP and UFIP **consists of records**, each with a header, supplying its type and length. These records are grouped into a buffer, to improve performance **by avoiding a call to SIDI or SOCKI for each record**. When the SCSI hardware is used, the buffer built by GFIP or UFIP is encapsulated by SID or IO-MAN respectively, into GCOS Stream Messages and/DBSP Stream Messages. These stream messages are described hereinbelow. When the sockets interface is used, GFIP/UFIP buffers are encapsulated by a socket header.

Records have a common header with the following format shown below in Table 4.1.3:

Table 4.1.3

Record Format

Bytes	Width	Field
00-01	02	data size
02-03	02	record type
04-64003	<64004	Records

- Both 'data size' and 'record type' are integers with the most significant byte in the lowest addressed byte. The values in these two fields have maximum value of 65535.

- 'data size' is the number of bytes in the record; it includes itself (2 bytes), the record type (2 bytes), and the record data. The maximum value is 64004; this is arbitrarily set, the actual maximum could be 65152.

- 'record type' is an integer with values between 0 and 65535. Zero is not used. Values 32768 and higher are reserved for system types. Values from 1 to 32767 are user types (there is no mechanism provided for users to use this field).

-

Open Request

An **Open Request record** is sent from GCOS to UNIX when an application calls the X_ETL_OPEN procedure:

Table 4.1.4

Open Request Record Format

Bytes	Width	Field
00-01	02	Data size
02-03	02	Record Type
04-07	04	ETL-OPEN-ACTION
08-11	04	ETL-DATA-FORMAT
12-15	04	Length of path name
16-19	04	ETL-MAX-RESPONSE-TIME
20-23	04	ETL-RCV-REC-SIZE
24-63	40	RFU
64-127	64	UserID
128-xx	xx	Path Name (null terminated "C" string)

Userid is a NULL-terminated C string. Its maximum length is 63 characters, excluding the NULL-terminator. Some systems may have lower limits. It is the Userid assigned to the GCOS program that is making the ETL calls.

Pathname is a NULL-terminated C string. Its maximum length is 1024 characters, excluding the NULL-terminator. It is the same as the ETL-PATHNAME parameter except that trailing white space has been deleted and the NULL-terminator has been added.

If the UNIX server detects an error processing the open request record, it sends an error record to GCOS. Otherwise, it does not send a response record; i.e., there is no execute response record.

4.1.10 Data Record

The GCOS data record contains application data sent from GCOS to UNIX or from UNIX to GCOS.

Table 4.1.7

4.1.10 Data Record

Bytes	Width	Field
00-01	02	Data size
02-03	02	Record Type
04-xx	yy	Application Data

6. 5. EIS

5.1 GCOS 8 Cobol-85 API

This section specifies the GCOS 8 Cobol-85 interface for the Fast-ETL product. The interface provides a **record oriented connection between the programs** using it. The program on GCOS 8 acts as a client, and the Fast-ETL program on UNIX acts as a server.

5.1.3 OPEN

```
CALL "X_ETL_OPEN" USING        ETL-STATUS-RTN,
                               ETL-FILE-ID,
                               ETL-PARAMETER-BLOCK,
                               ETL-PATH-NAME.
```

This procedure is one of two (X_ETL_EXECUTE is the other) that establish a connection to a server on UNIX. This procedure establishes a connection to the UNIX file server (UFAP) using the **sockets interface**.

The connection established by this procedure is a uni-directional connection used to read or write UNIX files from a GCOS program. Subsequent READ, or WRITE procedures get data from or put data to the specified UNIX file. The direction supported by a connection (i.e. read or write) is determined by the ETL-OPEN-ACTION parameter. Each invocation of this procedure starts a new instance of the file server, and the UNIX file specified will be created or accessed.

The "X_ETL_OPEN" procedure blocks execution until a connection is established or an error occurs.

ETL-STATUS-RTN

ETL-STATUS-RTN is an output parameter used to determine the success of this procedure. When successful, this procedure sets ETL-PRIMARY-STATUS to zero. If an error occurs, ETL-PRIMARY-STATUS is set to a non-zero value. The ETL-IMMEDIATE and ETL-ORIGINAL fields identify the error.

ETL-FILE-ID

ETL-FILE-ID is an output parameter that is meaningful only when ETL-STATUS-RTN reports success. It identifies the connection to the server, and it is an input parameter on subsequent procedures (e.g. X_ETL_READREC or X_ETL_WRITEREC) using this connection.

ETL-PARAMETER-BLOCK

ETL-PARAMETER-BLOCK is an input parameter containing several fields:

The ETL-IPADDRESS and ETL-PORT values identify the machine to be accessed. If ETL-PORT is zero (low-value), the default port is used.

ETL-MAX-RESPONSE-TIME specifies the number of seconds that GFIP allows sockets to wait before sockets must return with a timeout status. This time value applies for the duration of the connection; i.e., it applies to all subsequent ETL calls for the connection.

ETL-OPEN-ACTION must contain one of the following values:

ETLWRITE - The client is writing records; i.e., it is sending records to the UNIX server. "X_ETL_READREC" calls are not allowed for this connection.

ETLREAD - The client is reading records; i.e., it is receiving records from the UNIX server. "X_ETL_WRITEREC" calls are not allowed for this connection.

ETL-DATA-FORMAT must contain one of the following values:

ETLASCIIRecords sent over this connection are formatted as a standard text file record for the node that is receiving them. Records sent by GCOS have all trailing white space characters stripped from them. Records received by GCOS also have no trailing white space characters.

This format is convenient when transferring standard text files, such as source files or comma-delimited data files. Neither application needs to be aware of the file formats used by the other node.

ETLRAW Records sent over this connection are transferred as 8-bit characters. When received by GCOS, each 8-bit character is put into a 9-bit byte. No other changes are made. With the possible exception of the last record read, the length returned by the X_ETL_READREC function is the value specified by ETL-RCV-REC-SIZE.

This format is typically used by a GCOS application that needs to construct a non-text file on UNIX, and needs to be able to specify exactly what will be in the UNIX file. The sending application needs to be aware of the exact details of the other nodes file formats for this mode.

ETLBITS Records sent over this connection are transferred as a bit stream. Every 8 bytes from GCOS occupies 9 bytes in a UNIX buffer. Except for the bit padding that occurs when the record sent or received by GCOS is not a modulo 8 value, no change is made to the data.

With the possible exception of the last record read, the length returned by the X_ETL_READREC function is the value specified by ETL-RCV-REC-SIZE.

This format is useful for saving an arbitrary GCOS file on UNIX. It generally won't be usable on UNIX, but it can be transferred back intact to GCOS.

ETL-RCV-REC-SIZE specifies the size in bytes of the records that X_ETL_READREC returns when both of the following conditions are true:

1. ETL-OPEN-ACTION contains ETLREAD.
2. ETL-DATA-FORMAT contains either ETLRAW or ETLBITS.

ETL-PATHNAME

ETL-PATHNAME is an input parameter. This parameter is passed to the server and identifies a UNIX file. This parameter is processed to be convenient for the server to use as a pathname. Specifically, trailing white space characters are stripped.

The pathname may be either an absolute or relative pathname. Relative pathnames are relative to the home directory of the server process spawned by ETL's UNIX software. The home directory is derived from the USERID (account) name associated with the GCOS program performing the X_ETL_OPEN call. The USERID is converted to lower case in deriving the home directory.

If the ETL-OPEN-ACTION parameter is ETL-RECV, the file referenced by ETL-PATHNAME is accessed for reading. If ETL-SEND is specified, the file is created when it doesn't exist, and it is overwritten when it does exist.

5.1.7 WRITE A RECORD

```
CALL "X_ETL_WRITEREC" USING ETL-STATUS-RTN,
                              ETL-FILE-ID,
                              ETL-REC-LENGTH,
                              ETL-RECORD.
```

This procedure sends a record to the UNIX server program. X_ETL_WRITEREC moves the record located in 'ETL-record' into a collection buffer for the connection. The collection buffer is sent to the server when one of the following occurs:

- no more records can fit in the buffer,
- one of the following ETL functions is called. These functions automatically flush the buffer:

  a) X_ETL_CLOSE
  b) X_ETL_CHECKPOINT
  c) X_ETL_TERMINATE
  d) X_ETL_READREC

The number of bytes sent from 'ETL-record' is specified by the ETL-REC-LENGTH parameter.

This procedure does not normally block execution. X_ETL_WRITEREC returns to the caller after placing the record in a collection buffer. If the buffer is full and cannot be written (e.g. the server is not reading records for some reason), this function waits until one of the following occurs:

- the buffer can be written,
- a timeout occurs (time limit is specified by ETL-MAX-RESPONSE-TIME),
- some other error occurs.

ETL-STATUS-RTN

ETL-STATUS-RTN is an output parameter used to determine the success of this procedure. When successful, this procedure sets ETL-PRIMARY-STATUS to zero. If an error occurs, ETL-PRIMARY-STATUS is set to a non-zero value. The ETL-IMMEDIATE and ETL-ORIGINAL fields identify the error.

ETL-FILE-ID

ETL-FILE-ID is an input parameter that identifies the connection to the server. The value of ETL-FILE-ID was returned by the call to X_ETL_OPEN or X_ETL_EXECUTE that established the connection.

ETL-REC-LENGTH

ETL-REC-LENGTH is an input parameter whose value specifies the number of bytes of data to be sent to the server. The maximum value that ETL-REC-LENGTH may contain is 64,000. ETL-REC-LENGTH may contain a value of zero.

ETL-RECORD

ETL-RECORD is an input parameter that contains the data to be sent to the server. Unlike most of the parameters used for ETL calls, this parameter is NOT included in the COBOL copy file 'ETL_DATA_H'.

5.1.8 READ A RECORD

```
CALL "X_ETL_READREC" USING        ETL-STATUS-RTN,
                                  ETL-FILE-ID,
                                  length of ETL-buffer,
                                  ETL-buffer,
                                  ETL-RETURNED -LENGTH
                             [ON EXCEPTION  imperative-statement]..
```

This procedure gets a record sent by the server program. The maximum number of bytes that may be returned is specified by the 'length of ETL-buffer' parameter. The number of bytes returned in the record is in ETL-RETURNED-LENGTH. A return length value of zero bytes means that the server sent a zero-length record.

If a record is not available, execution is blocked until the server program returns a record or until a timeout/error occurs.

This procedure returns the status ETLEOF (end-of-file) only after all of the records sent by the server program have been read. Since X_ETL_READREC also generates an exception condition if and only if it returns the end-of-file status, GCOS programs may supply the optional 'ON EXCEPTION' clause to control the processing of end-of-file.

ETL-STATUS-RTN

ETL-STATUS-RTN is an output parameter used to determine the success of this procedure. When successful, this procedure sets ETL-PRIMARY-STATUS to zero. If an exception or an error occurs, ETL-PRIMARY-STATUS is set to a non-zero value. The ETL-IMMEDIATE and ETL-ORIGINAL fields identify the exception or error.

ETL-FILE-ID

ETL-FILE-ID is an input parameter that identifies the connection to the server. The value of ETL-FILE-ID was returned by the call to X_ETL_OPEN or X_ETL_EXECUTE that established the connection.

length of ETL-buffer

The 'length of ETL-buffer' is an input parameter whose value specifies the maximum number of bytes of data to be received from the server. Unlike most of the parameters used for ETL calls, this parameter is NOT included in the COBOL copy file 'ETL_DATA_H'. Instead, the length must either be placed in a variable of the program's own choosing or it may be specified by the COBOL phrase 'length of ETL-buffer', where the name 'ETL-buffer' is the name of the buffer used to receive the record.

ETL-buffer

ETL-buffer is an output parameter that contains the record received from the server. Unlike most of the parameters used for ETL calls, this parameter is NOT included in the COBOL copy file 'ETL_DATA_H'.

If ETL-buffer is too small to contain the record that was received, then X_ETL_READREC returns the amount of data that does fit in ETL-buffer, discards the remainder of the record, and returns the exception status of ETLBUFSIZE.

ETL-RETURNED-LENGTH

ETL- RETURNED-LENGTH is an output parameter that contains length in bytes of the record received from the server (in bytes).

5.1.11 DEFINE RECORDS

```
CALL "X_ETL_DEFINEREC" USING     ETL-STATUS-RTN,
                                           ETL-FILE-ID,
                                           ETL-PATHNAME.
```

This procedure assigns a record definition to the connection identified by ETL-FILE-ID. The record definition specifies the **data conversions that subsequent X_ETL_READREC and X_ETL_WRITEREC procedures perform**.

The specified GCOS file is read to obtain a record definition. The record definition **conforms** to that described in the RDBC Data Warehouse manuals available from Assignee. The textual record definition is parsed and used to define the conversions that read and write will use.

7. 6.0 SID Interface

6.1 Send Message Interface

GFIP calls SID's **Send Message function** to send a block of data to a DBSP that provides the Fast-ETL service. The Send Message interface is similar to the interface to SID's Pass Messages function, which is the function originally defined for exchanging data with a DBSP. The interfaces differ primarily in that the Send Message interface does not provide for an input buffer parameter. The Send Message call is shown below.

8. APPENDIX A
Data Transfer Request (DTR) File -- for use with X_ETL_DEFINEREC

```
/* Miscellaneous notes:
* 1. "-INDICATORS YES" implies that the number of Indicator Bits is
* the number of <FieldDesc> fields contained in the RECORDFORMAT command.
*/

/*-----------------------------------------------------------*/

<RecFormatCmd>::=   RECORDFORMAT <FieldDesc> [,<FieldDesc>...] ;

<FieldDesc>::=      <fieldname>  ( <datatype> )

<fieldname>::=      /* alpha-numeric string indicating name of record field */

<datatype>::=       BYTE (n) | BYTEINT | CHAR (n) | DATE | DECIMAL (x)
        DECIMAL (x,y) | FLOAT | GRAPHIC (n) | INTEGER | LONG VARBYTE |
        LONG VARCHAR | LONG VARGRAPHIC | SMALLINT | VARBYTE (n) |
        VARCHAR (n) | VARGRAPHIC (n)

/* Miscellanous notes:
* 1. Data types LONG VARBYTE and LONG VARCHAR imply a maximum
* length of 64000 1-byte elements (64000 bytes).
* 2. LONG VARGRAPHIC implies a maximum length of 32000 2-byte
* elements (64000 bytes).
* 3. All tokens comprising an individual <FieldDesc> must be contained on
* the same line.
*/
```